Proceedings of the IASTED International Conference
COMPUTATIONAL AND SYSTEMS BIOLOGY
November 13–14, 2006, Dallas, TX, USA
ISBN CD: 0-88986-601-5

A FAST ALGORITHM FOR SORTING BY SHORT SWAP

Xuerong Feng†, Ivan Hal Sudborough‡ and Enyue Lu§
† Department of Mathematics and Computer Science, Valdosta State University, Valdosta, GA 31698, USA
‡ Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083-0688, USA
§ Department of Mathematics and Computer Science, Salisbury University, Salisbury, MD 21801, USA

ABSTRACT
A short swap is an operation that switches two elements that have at most one element in between. In this paper, we consider the problem of sorting an arbitrary permutation by short swaps. We give an algorithm which sorts any permutation of length n within $(3/16)n^2 + O(n \log n)$ steps, improving the previous $(5/24)n^2 + O(n \log n)$ upper bound.

KEY WORDS
Short Swap, Sorting, Algorithm, Gene Rearrangement

1. Introduction

Analysis of genome rearrangements is a fundamental task in computational biology. Any genome rearrangement study involves solving a combinatorial puzzle of transforming one genome into another [1]. The shortest, also called the most parsimonious rearrangement scenarios represent the most likely molecular evolution path. Scientific studies show that for uni-chromosomal genomes, inversions (reversals) are the dominant rearrangement event, and for multi-chromosomal genomes, reversals, transpositions and translocations are common rearrangement events.

Research comparing human and mouse genomes shows that a large number of micro-rearrangements, *e.g*, intra-chromosomal rearrangement with a span < 1Mb, happen during the draft of human and mouse genomic sequence [2]. If we think of a genome as a particular ordering of genes, then the problem of analysis of rearrangement events, where one species has evolved into another, can be viewed as a problem of transforming one permutation into another by reversals. And if we take the micro-rearrangement scenario into account, the problem can be viewed as transforming one permutation into another by bounded length reversals.

So far, the majority of research has focused on calculating the unbounded reversal distance. Using the notion of *breakpoint*, Kececioglu and Sankoff [3] were the first to give a 2-approximation algorithm for sorting by unbounded length reversals. Pevzner [4], Christie [5] improved the approximation ratio, currently the best is a 1.375 approximation algorithm due to Berman [6]. Capara [7] proved this problem is NP hard. One also consider the case where each element of the permutation has either a "+" or "-" sign and when reversed, the sign changes from "+" to "-" or from "-" to "+". This is the case of *signed permutations*. Hanehalli and Pevzner [8] showed that sorting signed permutations in the minimum number of steps by unbounded length reversals is polynomial solvable. Kaplan, Shamir and Tarjan [9] gave a faster and simpler algorithm for this problem. For the unsigned permutation, if each reversal must start from the first element, the sorting problem is known as the *pancake problem* [10]. Gates and Papadimitriou showed an upper bound of $(5/3)n+5/3$ and a lower bound of $(17/16)n$. Heydari and Sudborough [11] tightened the lower bound to $(15/14)n$ and showed that the conjectured hardest signed permutation, namely $I_n = -1, -2, \ldots, -n$, can be sorted in $(3/2)n+1/2$ steps. Chen and Skiena [12] gave big-Oh asymptotic upper and lower bounds for sorting with fixed length reversals. Walter, Dias and Meidanis [13] discuss the problem of sorting by reversals and simultaneous transpositions. Gu, Peng and Sudborough gave an approximation algorithm for this problem [14]. Lin and Xue [15] discuss the signed version of sorting by both reversal and transposition.

The problem of sorting an arbitrary permutation by switching two adjacent elements or two elements with one element in between is called sorting by *short swap* [16]. We can view a short swap operation as a substring reversal of length 2 or 3. Heath and Vegera [16] showed a $(1/4)n^2$ upper bound and a $(1/6)n^2$ lower bound for this problem. In [17] we gave an algorithm with an improved upper bound of $(5/24)n^2 + O(n \log n)$. In this paper, we further improve the upper bound and show a $(3/16)n^2 + O(n \log n)$ upper bound.

In section 2, we give the proof of the new upper bound, in section 3, we discuss some open questions.

2. A $(3/16)n^2 + O(n \log n)$ Upper Bound

Let $L = \{1, 2, 3, , n\}$. A permutation $\pi = \pi_1\pi_2\pi_3\ldots\pi_n$ of L is an ordered arrangement of the elements in L. The element at position i is denoted by $\pi(i)$, where $1 \leq i \leq n$. We will use integers $1, 2, \ldots, n$ to indicate both positions and elements. Without loss of generality, assume $n \equiv 1 \pmod 4$, let $n = 4k+1$ and let k be even. First we use a linear number of steps to put element $2k+1$ in the middle, *i.e.* at position $2k+1$. This divides the permutation into a left and right part, where each part has $2k$ numbers. Assume there are p numbers ($0 \leq p \leq 2k$) which are in the range $[1, 2k]$ but are positioned in the right part, there must also have p numbers which are in the range $[2k+2, 4k+1]$ but are positioned in the left part. Based on p's value, we consider the following two cases and for each case, we use a different algorithm to sort.

2.1 Case 1: $0 \leq p \leq k$

For this case, independently for the left and right part, we do the following. For the left part, without sorting we move the p numbers in the range $[2k+2, 4k+1]$ to the middle, occupying positions from $2k-p+1$ to $2k$. Similarly for the right part, without sorting we move the p numbers in the range $[1, 2k]$ to the middle, occupying positions from $2k+2$ to $2k+p+1$. During above procedure, we use length 3 reversals as much as possible. Use the following permutation as one example:

Position i :	1	2	3	4	5	6	7	8	9	10	11	12	13
π:	$\underline{9}$	5	4	1	$\underline{13}$	3	7	11	10	$\underline{6}$	12	8	$\underline{2}$

here $n = 13$, $k = 3$ and $p = 2$, since number 9, 13 which are in the range $[2k+2, 4k+1]$ but are positioned in the left part on position 1 and 5. There must also have two numbers, in above case, 6 and 2, which are in the range $[1, 2k]$ but are positioned in the right part. Consider the left part only, by a length 2 reversal, we switch 13 with 3 and get the following permutation:

Position i :	1	2	3	4	5	6	7	8	9	10	11	12	13
π^1:	$\underline{9}$	5	4	1	3	$\underline{13}$	7	11	10	$\underline{6}$	12	8	$\underline{2}$

Next, by two length 3 reversals, number 9 first switches with number 4, then switch with number 3, we get the following permutation:

Position i :	1	2	3	4	5	6	7	8	9	10	11	12	13
π^2:	4	5	3	1	$\underline{9}$	$\underline{13}$	7	11	10	$\underline{6}$	12	8	$\underline{2}$

We do similar thing on right part and move number 6 and 2 to position 8 and 9 respectively and get the following permutation:

Position i :	1	2	3	4	5	6	7	8	9	10	11	12	13
π^3:	4	5	3	1	$\underline{9}$	$\underline{13}$	7	$\underline{6}$	$\underline{2}$	11	10	8	12

Next, first we switch number 13 with 2 by a length 3 reversal, followed by a length 2 reversal. We switch 9 with 6 first by a length 2 reversal, followed by a length 3 reversal, and the permutation now is:

Position i :	1	2	3	4	5	6	7	8	9	10	11	12	13
π^4:	4	5	3	1	$\underline{6}$	$\underline{2}$	7	$\underline{9}$	$\underline{13}$	11	10	8	12

We do similar operations on an arbitrary permutation satisfying the condition stated in case 1, namely, $0 \leq p \leq 2k$. For a general case, the worst case happens when the $2p$ numbers are at the two ends. That means for the left part, the p numbers which are in the range $[2k+2, 4k+1]$ occupy positions from 1 to p, and similarly for the right part, the p numbers in the range $[1, 2k]$ occupy positions from $4k-p+2$ to $4k+1$. In this case, it takes at most $2 * p * [(n/2 - p)/2]$ steps to move them to the middle. Once these $2p$ numbers are put into the middle, we swap the left p numbers with the right p numbers. Similarly as we show in above example, this is done in p phases, at phase i ($1 \leq i \leq p$), using length 2 or 3 reversal, we put the number at position $2k+i+1$ into position $2k-p+i$. During this procedure, we use length 3 reversals as much as possible and only use length 2 at the beginning or at the end when necessary. Note that the p numbers originally occupying positions from $2k-p+1$ to $2k$ are pushed to the right and will then occupy positions from $2k+2$ to $2k+p+1$. It takes $p*(p/2)$ steps to do the swaps. Combining the above two steps, the total number of exchanges needed is:

$$\begin{aligned} &2 * p * [(n/2 - p)/2] + p*(p/2) \\ &= (1/2)np - p^2 + (1/2)p^2 \\ &= (1/2)np - (1/2)p^2 \qquad (1) \end{aligned}$$

Then all the numbers in the range $[1, 2k]$ are in the left part and all the numbers in the range $[2k+2, 4k+1]$ are in the right part. Once this is done, we treat the left and right part as two separate permutations each of size $(n-1)/2$, and we will apply our algorithm separately to sort the left and right part permutation. Notice function in (1) is an increasing function of p in the range of $[1, k]$. When $p = k$ ($k = (n-1)/4$), the function has the maximum value which is $(3/32)n^2$. Let $T(n)$ denote the total steps needed to sort a permutation π, if every time case (1) happens, we have the following recursive formula:

$$T(n) = 2*T(n/2) + (3/32)n^2 + O(n)$$

The solution to above recurrence is $T(n) = (3/16)n^2 + O(n \log n)$

2.2 Case 2: $k < p \leq 2k$

For this case, our algorithm will include 3 stages.

Stage 1: Normalization

For this stage, we use a similar algorithm as described in [17]. To make the analysis simple, we consider an even integer n, suppose that the elements of the set $\{1,2, \ldots, n\}$ are partitioned into two subsets A and B, each with $n/2$ elements. For an arbitrary permutation π of L, where $L = \{1, 2, 3 \ldots n\}$, say $\pi = \pi_1\pi_2\pi_3\ldots\pi_n$, we consider the problem of moving all of the elements in A to the odd positions, namely position 1, 3, 5, …, n-1 and all of the elements of B to the even positions, namely position 2, 4, 6, …, n. Neither the elements of A nor the elements of B need to be sorted. We call a procedure to do this *normalization.* For example, the set A could be the largest $n/2$ elements and the set B could be the smallest $n/2$ elements and, for $n = 12$, one could have the permutation $\pi = (1, 12, 2, 11, 3, 10, 4, 9, 5, 8, 6, 7)$. Then, every element of A $=\{7,8,9,10,11,12\}$ is in an even position and every element of B $= \{1,2,3,4,5,6\}$ is in an odd position, so all are out of place. In this case, simply exchanging each element in an odd position with its neighboring even position element, resulting in the permutation (12, 1, 11, 2, 10, 3, 9, 4, 8, 5, 7, 6) accomplishes the goals of normalization.

We now establish, for any choice of a partition of $\{1,2, \ldots, n\}$ into two sets A and B, each with $n/2$ elements, and any permutation π of $\{1,2, \ldots, n\}$ the worst case number of short swaps to do the task of normalization. That is, we describe an optimum normalization procedure and analyze the number of steps the procedure takes. The procedure begins with a left-to-right scan of the given permutation π. Call the elements in A that are in even positions and the elements of B that are in odd positions *out of place* elements. Then, during this initial scan of the input, the procedure exchanges any out of place elements that are adjacent. Clearly, in general, out of place elements will still exist, but they will not be adjacent.

The remaining set of out of place elements can be put into positions of the correct parity by moving them to the positions of out of place elements of the other set through positions of elements that are not out of place by exchanges between elements at distance two. Note that the number of elements in A that are out of place is equal to the number of elements in B that are out of place, so if there is an out of place element from one set there must be a one in the other set. Now, consider a situation where we move the leftmost out of place A element, say x, in an even position, to the position of the leftmost out of place B element, say y, in an odd position, where y is to the left of x. As x and y are not adjacent, x will be swapped with other even position elements by length 3 reversals until it becomes adjacent with y, and then x and y will be exchanged. The elements in even positions that x is swapped with are not out of place, as x is the leftmost out of place element of A, so moving them to a different even position does not make them out of place. So, in this movement of the leftmost out of place A element to the position of the leftmost out of place B element, we eliminate both the leftmost out of place A element and the leftmost B element. The situation is entirely symmetric when x is to the left of y. In that case, one moves y to the left through other odd positions by length 3 reversals until y becomes adjacent to x.

The worst case is when n is a multiple of 4 and there are $n/4$ out of place B elements in positions $1,3,5, \ldots, (n/2)-1$ and there are $n/4$ out of place A elements in positions $(n/2)+2, (n/2)+4, \ldots, n$. In this case, each of the $n/4$ iterations takes the leftmost out of place A elements to the leftmost out of place B element, by the process already described, and takes $(n/4)+1$ exchanges. That is, at each iteration the leftmost out of place element of A is at a distance of $(n/2)+1$ from the leftmost out of place element of B, and as all but the last exchange is a reversal of length 3, the total number of reversals is $(n/4)+1$. Consequently, the total number of steps in the worst case is $(n/4)*(n/4+1) = (n/4)^2 + O(n) = n^2/16 + O(n)$.

Now consider the case 2, where $k < p \leq 2k$ and $k = (n-1)/4$. Independently for the left part, for all those p numbers in the range $[2k+2, 4k+1]$, using the normalization procedure described above, we place the biggest k of them on the left even positions. Similarly for the right part, of all the p numbers in the range $[1, 2k]$, we place the smallest k of them on the right even positions. This takes $2*(1/16)(2k)^2 = (1/2)k^2 + O(k) = (1/32)n^2 + O(n)$, since $k = (n-1)/4$ steps. Notice that after the above steps, there are $p-k$ numbers, which are in the range $[2k+2, 4k+1]$, which have been put in the left odd positions, and there are $p- k$ numbers, which are in the range $[1, 2k]$, which have been put in the right odd positions.

Stage 2: Swap

Now consider the biggest k numbers, which are $3k+2, 3k+3, 3k+4, \ldots, 4k+1$ since we assume $n = 4k+1$ and k is even. After Stage 1 normalization procedure, some of these k numbers are put on the left even positions. The remaining are put on right odd positions. Similarly, for the smallest k numbers, after stage 1, some are put on right even positions. The remaining are put on left odd positions. We swap the numbers on left even positions with those on right even positions.

This is done in k phases, at phase i ($1 \leq i \leq k$), if the element at position $2k-2i+2$ belongs to the range $[3k+2, 4k+1]$, we use length 3 reversal to move it right to the position $4k-2i+2$. We do this by swapping it with elements at position $2k-2i+4, 2k-2i+6, \ldots, 4k-2i+2$. This takes k steps. If an element at position $2k-2i+2$ does not belong to the range $[3k+2, 4k+1]$, let's denote it by "a". We shall swap "a" with elements at position $2k-2i+4$, $2k-2i+6$, …, etc. Suppose at some point we found it is at distance ≤ 2 from some element "b" which is in a right odd position and belongs to the range $[3k+2, 4k+1]$. In that case, we use an exchange to swap "a" with "b". Now "b" is on even position and we continue swapping "b" with elements on subsequent even positions until "b" is put into position $4k-2i+2$. In this case, it takes $k+1$ step to move an element from position $2k-2i+2$ to position $4k-2i+2$. If "a" does not pass any element which is on right odd position and belongs to the range $[3k+2, 4k+1]$, we simply move "a", until "a" is put in position $4k-2i+2$. This takes k steps.

As above movements take place, *i.e.* after each phrase i ($1 \leq i \leq k$), elements on the right even positions are moved 2 positions left. Symmetrically if the element originally on position $2k+2i$ belongs to the range $[1, k]$, after k phases it will be put on position $2i$. If it does not belong to the range $[1, k]$, we denote it as "c". Suppose at phase i, it passes an element "d" which is on left odd position and belongs to the range $[1, k]$. Then we use an exchange to swap "c" with "d". Then "d" is in an even position and it will be shifted left 2 positions with each phase until it is put on position $2i$ after k phases. If the element originally on position $2k+2i$ does not belong to the range $[1, k]$ and during those k phases it does not pass over any elements which are on left odd positions and belong to the range $[1, k]$, it will simply be put into position $2i$. For stage 2, it takes $k * k = (1/16)n^2+O(n)$ steps of short swaps, since $k = (n-1)/4$.

Stage 3: Locally sort

We will use the same algorithm as used to sort the basic class permutation in [17] Stage 3. In brief, this is done in 2 steps:

Step 1: For the left part of the permutation, starting from element 1, and in increasing order, using reversals of length 2 or 3 and through odd positions, put 1, 2, 3, 4,…, $k/2$ into their correct positions. Similarly for the right part, starting from element $4k+1$, in decreasing order, using reversals of length 2 or 3 through the odd positions, put $4k+1$, $4k$, $4k-1$,…, $4k-k/2+2$ into their correct position. As a result of the above, elements in the range $[(k/2)+1, k]$ will be put together and will occupy positions from $(k/2)+1$ to k in random order. Similarly for the elements in the range $[3k+2, 4k-(k/2)+1]$, they will be put together and will occupy positions from $3k+2$ to $4k-(k/2)+1$, but in random order.

Step 2: Now, independently for the left and right part, using the earlier swap sorting algorithm described in [17], sort the elements that occupying position from $(k/2)+1$ to k and from $3k+2$ to $4k-(k/2)+1$. As described in [17], the above procedure needs a total of $(3/64)n^2+O(n)$ steps. As a result, the elements in the range $[1, k]$ and $[3k+2, 4k+1]$ are sorted. Note, in [17], for the basic class permutation, after Stage 2, numbers in the range $[1, k]$ are all on left even positions and numbers in the range $[3k+2, 4k+1]$ are all on right even positions. But for an arbitrary permutation π of length $4k+1$ which satisfy $k < p \leq 2k$, after stage 2, all the elements in the range $[1, k]$ are in the left part of the permutation, but might not all be on even positions, some may be on the left odd positions; similarly for the right part, all the elements in the range $[3k+2, 4k+1]$ are in the right part of the permutation but might not all be on the even positions, some may be on the right odd positions. Below we show that for an arbitrary permutation π which satisfy $k < p \leq 2k$, using the above algorithm, it takes equal or less than $(3/64)n^2+O(n)$ steps to get the numbers in the range $[1, k]$ and $[3k+2, 4k+1]$ into sorted order.

Let us use π_r and π_l to denote the left part and right part of the permutation after Stage 2. As the left part and right part are symmetric, we shall just consider the right part first. For the right part of the permutation, from left to right, let us use $a_1, a_2, a_3,…, a_j$ $(0 \leq j \leq k)$ to denote those elements that are on the right even positions but do not belong to the range $[3k+2, 4k+1]$. Among them a_j is the rightmost. There must be j elements which are in the range $[3k+2, 4k+1]$ but are on right odd positions, let use $b_1, b_2, b_3, …, b_j$ $(0 \leq j \leq k)$ to denote them, where b_1 is the leftmost. We claim that b_1's position must be to the right of a_j's position. This is the result of Stage 2. According to our algorithm in Stage 2, if b_1's position were to the left of a_j's position, then a_j would pass through b_1 and the two would get exchanged. Below Figure 1 describe the right part of the permutation after stage 2. We use $\square$ to denote the odd position and "_" to denote the even position.

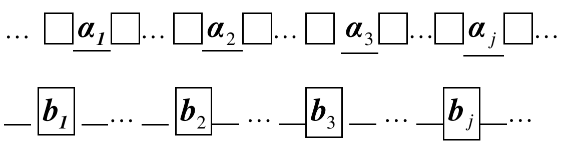

Figure 1. Permutation π_r

We construct a basic class permutation δ_r relevant to the permutation π_r, we switch a_1 with b_1, a_2 with b_2, …, and a_j with b_j and all the other elements remain unchanged. See Figure 2 for the resulting permutation δ_r.

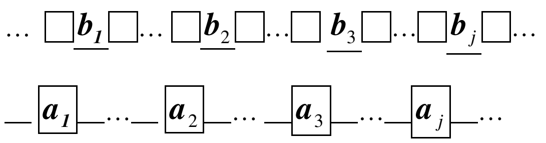

Figure 2. Permutation δ_r

For permutation δ_r, all the numbers in the range $[3k+2, 4k+1]$ are on even positions. Now compare permutation π_r with δ_r, notice that in π_r, b_1 through b_j are more close to the right end than they are in δ_r, as b_1 through b_j belong to the range $[3k+2, 4k+1]$, that means it takes less steps to put them into their correct positions in π_r than in δ_r. Similar analysis applies to the left part of the permutation, suppose we get δ_l from π_l, it will take less steps in π_l than in δ_l to put the numbers in the range $[1, k]$ into their correct positions. We know that δ_l and δ_r can be sorted within $(3/64)n^2 +O(n)$ steps by using the algorithm described in [17] stage 3, so π_l together with π_r will use equal or less than $(3/64)n^2+O(n)$ steps to get the numbers in the range $[1,k]$ and $[3k+2, 4k+1]$ sorted.

After the above 3 stages, numbers in the range $[1, k]$ and $[3k+2, 4k+1]$ are sorted and the middle is a permutation of length $2k+1$, which we will sort it recursively by apply the algorithm describe above. Since in normalization stage, it takes at most $(1/32)n^2+O(n)$ steps, in swap stage, it takes at most $(1/16)n^2+O(n)$ steps, and in locally sort stage, it takes at most $(3/64)n^2+O(n)$ steps. Let $T(n)$ denote the total steps needed to sort a permutation π in the above case 2. If we assume that every time case (2) is true, we have the following recursive formula:

$$T(n) = T(n/2) + (1/32)n^2+ (1/16)n^2+ (3/64)n^2+ O(n)$$
$$= T(n/2) + (9/64)n^2+ O(n)$$

The solution to above recurrence is

$$T(n) = (3/16)n^2+O(n \log n)$$

Combine the recursion we get for case 1 and case 2, we have the following:

$$T(n) = \begin{cases} 2T(n/2) + (3/32)n^2 + O(n) & (0 \le p \le k) \\ T(n/2) + (9/64)n^2 + O(n) & (k < p \le 2k) \end{cases}$$

And using induction on n and on case 1 and case 2, we conclude that:

$$T(n) \le (3/16)\, n^2 + O(n \log n)$$

This yields our improved upper bound for sorting by short swap.

3. Conclusion and Open Questions

We have given a recursive algorithm that sorts any permutation by short swaps within $(3/16)n^2$ +O(n log n) steps. Some questions remain. Can one improve the $(3/16)n^2$ +O(n log n) general upper bound? What is the worst case permutation for sorting by short swaps? Can one get a better approximation algorithm for sorting by short swaps?

For the first question, the answer is positive. Recall that in Section 2, for Case 2, we obtained the following recursive formula: $T(n) = T(n/2) + (9/64)n^2 +$ O(n). In this recurrence, out of $(9/64)\ n^2$, $(3/64)\ n^2$ represents the number of steps used for Stage 3 and Step 2, which was described in detailed in [17]. In order to sort the elements in the range $[(k/2)+1, k]$ and $[3k+2, 4k-(k/2)+1]$, we use the algorithm in [16]. The algorithm in [16] has an upper bound of $(1/4)n^2$+ O(n). If we use the improved $(5/24)n^2$+ O(n log n) upper bound in [17], and do not differentiate between Case 1 and Case 2. That is, use the algorithm described for Case 2 in Section 2 for every permutation π, even if $0 \le p \le k$. If $0 \le p \le k$, simply spread the p numbers on left/right even positions and swap them. By doing this, we are able to get an upper bound which is better than $(3/16)\ n^2$ +O(n log n). However, it is a very small improvement and hence is not included in the results given here.

For the second question, the backwards order permutation seems like a good candidate for a worst case permutation, since it has the maximum number of inversions, namely $n(n-1)/2$. But we have shown in [17] that we can sort it in optimum time, namely in $(1/6)n^2$ +O(n log n) steps. Let B be the set of permutation that all elements in the left half are bigger than all elements in its right half. In [17], we call set B the basic class permutations. All permutation in B has the maximum vector sum of $(n^2)/2$, we have shown that this class of permutations can be sorted in $(3/16)n^2$ + O(n log n) steps. Is the class B permutations hardest to sort? If so, we need to know how to prove it.

For the third question, currently, there is a 2-approximation algorithm due to Heath and Vergara [16]. We have shown a 3/2-approximation algorithm for all permutations in the basic class B in [17].

References

[1] H. J. Greenberg, W. E. Hart, G. Lancia, "Opportunities for Combinatorial Optimization in Computational Biology", *INFORMS Journal of Computing*, 2003

[2] P. A. Pevzner, G. Tesler, "Genome Rearrangements in Mammalian Evolution: Lessons From Human and Mouse Genomes", *Genomes*, 2003, Jan; 13(1):37~45

[3] J. Kececioglu, D. Sankoff, "Exact and Approximation Algorithm for Sorting by Reversals, with Application to Genome Rearrangement", *Algorithmica*, 13:1/2, 1995, pp. 180-210.

[4] V. Bafna, P.A. Pevzner, "Genome Rearrangement and Sorting by Reversals", *SIAM Journal of computing*, Vol. 25, No. 2, 1996, pp.272-289.

[5] D.A. Christie, "A 3/2-approximation Algorithm for Sorting by Reversals", *Proceedings of the Ninth Annual ACM-SIAM on Discrete Algorithm*, 1998, pp.244 – 252.

[6] P. Berman, S. Hannenhalli, M. Karpinski, "1.375-approximation Algorithm Sorting by Reversals", in *Proceedings of Annual European Symposium on Algorithm (ESA),* volume 2461 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 200-210.

[7] A. Caprara, "Sorting Permutations by Reversals and Eulerian Cycle Decompositions", *SIAM Journal of Discrete Mathematics*, Vol. 12, No. 1, 1999, pp. 91-110.

[8] S. Hannehalli, P.A. Pevzner, "Transforming Cabbage into Turnip (Polynomial Algorithm for Sorting Signed Permutations by Reversals)", in *Proc. 27th Annual ACM Symposium on the Theory of Computing*, ACM Press, New York, 1995, pp.178-187.

[9] H. Kaplan, R. Shamir, R.E. Tarjan, "A faster and Simpler Algorithm for Sorting Signed Permutation by Reversals", *SIAM Journal of Computing*, Vol. 29, No.3, 1999, pp.880-892.

[10] W.H. Gates, C.H. Papadimitrioro, "Bounds for Sorting by Prefix Reversal", *Discrete Mathematics*, 27, 1979, pp. 47-57.

[11] M.H. Heydari, I.H. Sudborough, "On sorting by Prefix Reversals and the Diameter of Pancake

Network", *Journal of Algorithm*, 25(1), 1997, pp.67-94.

[12] T. Chen, S.S. Skiena, "Sorting with fixed-length reversals", *Discrete Applied Mathematics*, 71 (1996), pp. 269-295

[13] M.E. Walter, Z.Dias and J. Meidanis, "Reversal and Transposition Distance of Linear Chromosomes", *Proceedings of SPIRE'98 - String Processing and Information Retrieval: A South American Symposium*. September, 9-11, 1998, pp. 96-102.

[14] Q.P. Gu, S. Peng, "Approximation Algorithm for Genome Rearrangements by Reversals and Transpositions", *Theoretical Computer Science*, 210(2), 1999, pp.327-339.

[15] G.H. Lin, G. Xue, "Signed Genome Rearrangements by Reversals and Transpositions: Models and Approximations", *Theoretical Computer Science*, 259(1-2), 2001, pp.513--531.

[16] L.S. Heath, J.C. Vergara, "Sorting by Short Swaps", *Journal of Computational Biology*, 2003: 10(5), pp.75-89.

[17] X.Feng, Z. Meng, I.H.Sudborough, "Improved Upper Bound for Sorting by Short Swaps", *IEEE Symposium on Parallel Architectures, Algorithms and Networks*, 2004, pp. 98-103.